Estruturas de Dados - ESP412

Profª Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal de Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

December 5, 2024

Listas, Filas e Pilhas

Listas

Listas

Em ciência da computação, uma lista ou sequência é uma estrutura de dados abstrata que implementa uma coleção ordenada de valores, onde o mesmo valor pode ocorrer mais de uma vez.

Listas

Em ciência da computação, uma lista ou sequência é uma estrutura de dados abstrata que implementa uma coleção ordenada de valores, onde o mesmo valor pode ocorrer mais de uma vez.

Uma instância de uma lista é uma representação computacional do conceito matemático de uma sequência finita, que é, uma tupla. Uma lista é armazenada dinamicamente em memória RAM.

Listas

Uma lista possui algumas propriedades:

- Os nodos são criados dinamicamente, à medida que for necessário. Assim, a quantidade total de memória usada para a lista depende da quantidade de dados nela armazenados (compare isso com um vetor ou matriz).

Listas

Uma lista possui algumas propriedades:

- Os nodos são criados dinamicamente, à medida que for necessário. Assim, a quantidade total de memória usada para a lista depende da quantidade de dados nela armazenados (compare isso com um vetor ou matriz).
- A lista não precisa ocupar uma área de memória contígua: como nodos são alocados dinamicamente, eles podem ocupar áreas de memória arbitrárias, e não há nenhuma relação entre a localização dos nodos em memória e sua ordem na lista (novamente compare isso com um vetor ou matriz).

Listas

Uma lista possui algumas propriedades:

- Não é possível indexar os nodos, por isso para acessar um nodo deve-se obrigatoriamente procurá-lo a partir do início da lista, seguindo cada nodo até chegar àquele procurado.

Listas

Uma lista possui algumas propriedades:

- Não é possível indexar os nodos, por isso para acessar um nodo deve-se obrigatoriamente procurá-lo a partir do início da lista, seguindo cada nodo até chegar àquele procurado.
- Para adicionar um nodo, basta modificar a referência do nodo que o antecede na lista. Assim, não é necessário "empurrar" os nodos seguintes para frente (como seria o caso de um vetor).

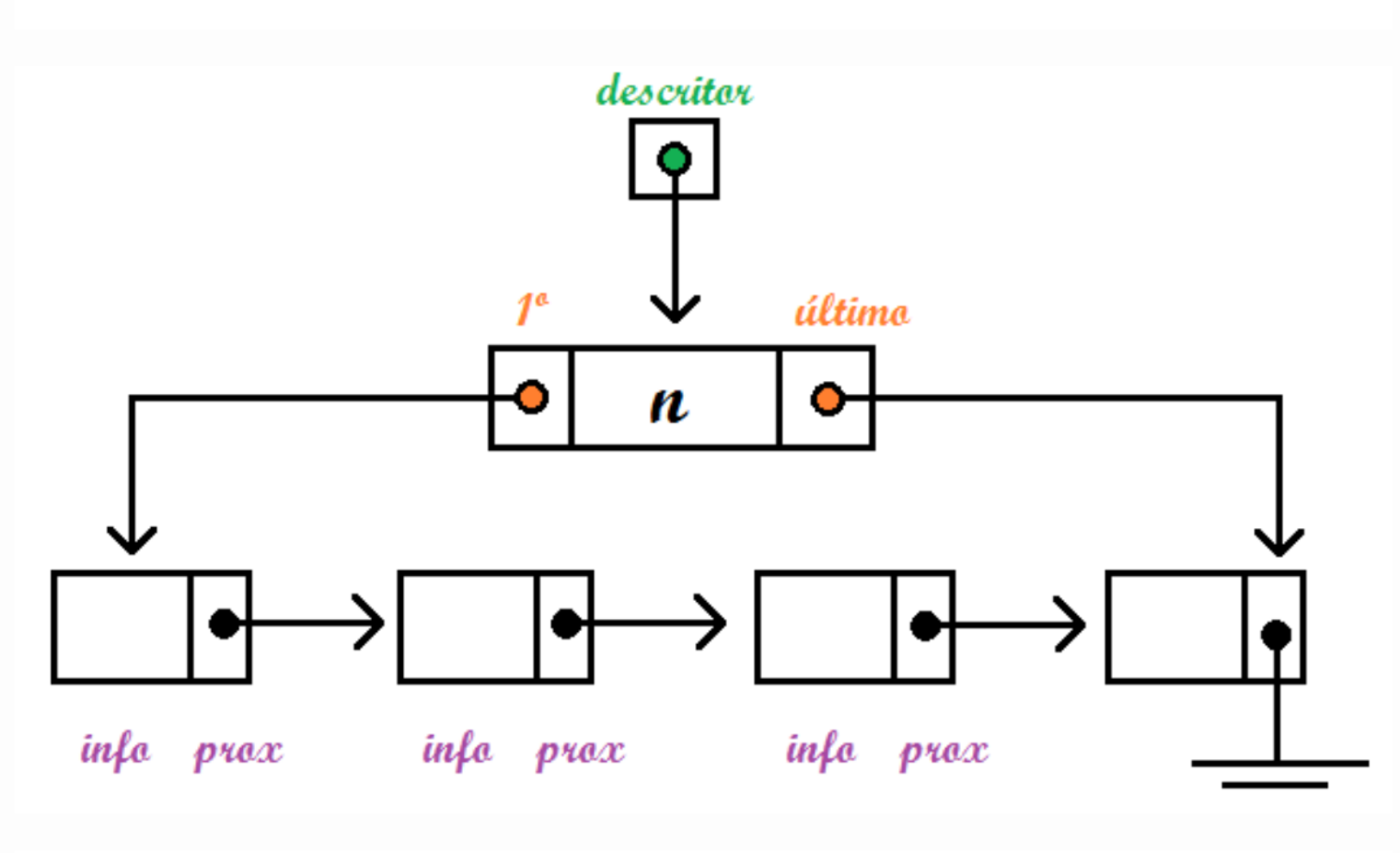
Listas

Uma lista possui algumas propriedades:

- Não é possível indexar os nodos, por isso para acessar um nodo deve-se obrigatoriamente procurá-lo a partir do início da lista, seguindo cada nodo até chegar àquele procurado.
- Para adicionar um nodo, basta modificar a referência do nodo que o antecede na lista. Assim, não é necessário "empurrar" os nodos seguintes para frente (como seria o caso de um vetor).
- Para remover um nodo é a mesma coisa: basta modificar a referência de seu nodo antecessor. Assim, não é necessário "deslocar pra trás" os nodos seguintes (como seria o caso de um vetor).

Listas

A lista possui um descritor que indica o nó inicial e o nó final da lista, além de indicar quantos elementos (n) contém a lista.



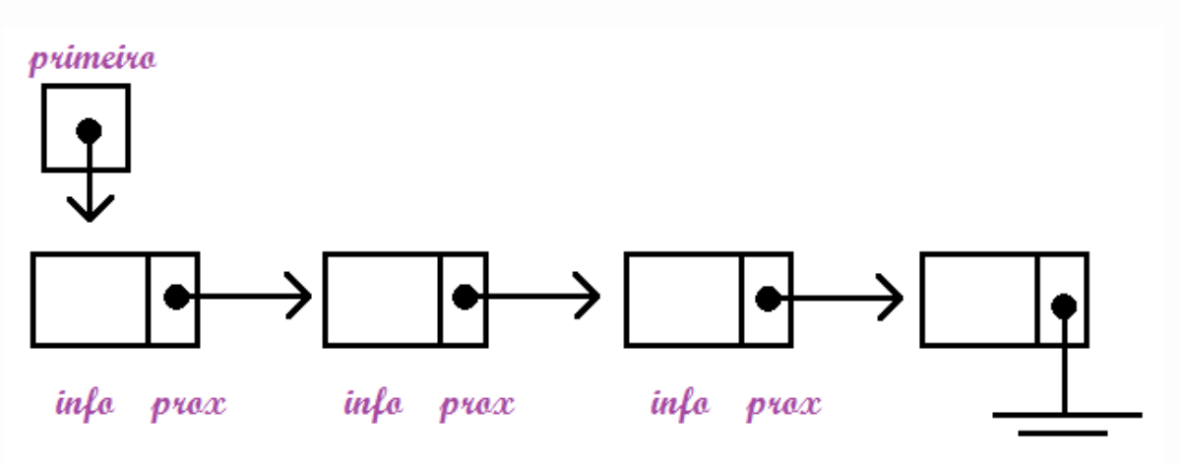
Listas Simplesmente Encadeadas

Listas Simplesmente Encadeadas

Uma Lista Simplesmente Encadeada é uma estrutura de dados em que os objetos estão organizados de forma linear, porém, diferente de um vetor, não é uma estrutura de dados indexada, é uma estrutura de dados em que os elementos são interligados por ponteiros. Desta forma, os nodos são encadeados de forma unidirecional, pode-se percorrer a lista do primeiro nodo em direção ao último nodo, mas não na direção contrária. [Cormen et al. 2009]

Listas Simplesmente Encadeadas

Uma Lista Simplesmente Encadeada é uma estrutura de dados em que os objetos estão organizados de forma linear, porém, diferente de um vetor, não é uma estrutura de dados indexada, é uma estrutura de dados em que os elementos são interligados por ponteiros. Desta forma, os nodos são encadeados de forma unidirecional, pode-se percorrer a lista do primeiro nodo em direção ao último nodo, mas não na direção contrária. [Cormen et al. 2009]



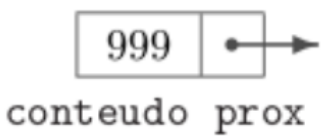
Listas Simplesmente Encadeadas

Cada nodo, célula ou elemento, da Lista Simplesmente Encadeada é composto pelo conteúdo do nodo da lista e por um ponteiro para o próximo nodo. Assim, uma Lista Simplesmente Encadeada é na verdade uma lista de registros. Em linguagem C, uma lista de *Structs*.

Listas Simplesmente Encadeadas

Cada nodo, célula ou elemento, da Lista Simplesmente Encadeada é composto pelo conteúdo do nodo da lista e por um ponteiro para o próximo nodo. Assim, uma Lista Simplesmente Encadeada é na verdade uma lista de registros. Em linguagem C, uma lista de *Structs*.

```
typedef struct reg {
    int          conteudo;
    struct reg *prox;
} celula;
```



Listas Simplesmente Encadeadas

Cada nodo, célula ou elemento, da Lista Simplesmente Encadeada é composto pelo conteúdo do nodo da lista e por um ponteiro para o próximo nodo. Assim, uma Lista Simplesmente Encadeada é na verdade uma lista de registros. Em linguagem C, uma lista de *Structs*.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct reg{
    int         conteudo;
    struct      reg *prox;
}celula;
```

Listas Simplesmente Encadeadas

Inserir:

O interessante de estruturas de dados que utilizam ponteiros é que elas não têm limites de tamanho. O limite é o tamanho da memória disponível para armazená-las. Cria-se o primeiro elemento, o topo, ou cabeça, e a partir deste, cresce a lista de acordo com a necessidade do usuário.

Listas Simplesmente Encadeadas

Inserir:

O interessante de estruturas de dados que utilizam ponteiros é que elas não têm limites de tamanho. O limite é o tamanho da memória disponível para armazená-las. Cria-se o primeiro elemento, o topo, ou cabeça, e a partir deste, cresce a lista de acordo com a necessidade do usuário.

```
celula *inserir(int x, celula *p){
    celula *nova;
     nova = malloc (sizeof (celula));
     nova->conteudo = x;
     nova->prox = NULL;
    if (p != NULL)
          nova->prox = p;
    p = nova;
    return p;
}
```

Listas Simplesmente Encadeadas

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa bem simples e pode ser feita tanto recursivamente como de forma iterativa. Vejamos as duas soluções possíveis:

Listas Simplesmente Encadeadas

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa bem simples e pode ser feita tanto recursivamente como de forma iterativa. Vejamos as duas soluções possíveis:

```
void imprimi_r(celula *listaEncadeada) {
    if (listaEncadeada != NULL) {
        printf ("%d\n", listaEncadeada->conteudo);
        imprimi_r(listaEncadeada->prox);
    }
}
void imprimir(celula *listaEncadeada) {
    celula *p;
    for (p = listaEncadeada; p != NULL; p = p->prox)
        printf ("%d\n", p->conteudo);
}
```

Listas Simplesmente Encadeadas

Inserir no FINAL:

Função INSERIR colocando os elementos sempre no final da lista

```
celula *inserir(int x, celula *p){
    celula *novo, *topo;
    topo = p;
    novo = malloc (sizeof (celula));
    novo->conteudo = x;
    novo->prox = NULL;
    if (p!=NULL){
        for(p;p->prox!= NULL;p=p->prox);
        p->prox=novo;
    }else topo = novo;
    return topo;
}
```

Listas Simplesmente Encadeadas

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse. Esta função também pode ser construída de forma iterativa ou recursiva, vejamos as duas opções:

Listas Simplesmente Encadeadas

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse. Esta função também pode ser construída de forma iterativa ou recursiva, vejamos as duas opções:

```
celula *buscar(int x, celula *listaEncadeada){
    celula *p;
    p = listaEncadeada;
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```

Listas Simplesmente Encadeadas

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse. Esta função também pode ser construída de forma iterativa ou recursiva, vejamos as duas opções:

```
celula *busca_r(int x, celula *listaEncadeada){
    if (listaEncadeada == NULL)  return NULL;
    if (listaEncadeada->conteudo == x)  return listaEncadeada;
    return busca_r (x, listaEncadeada->prox);
}
```

Listas Simplesmente Encadeadas

Remover:

A função remover pode ser construída de uma forma simples, onde passa-se o ponteiro, por valor, do elemento anterior e deseja-se remover o próximo elemento:

Listas Simplesmente Encadeadas

Remover:

A função remover pode ser construída de uma forma simples, onde passa-se o ponteiro, por valor, do elemento anterior e deseja-se remover o próximo elemento:

```
void remover(celula *p){
    celula *lixo;
    lixo = p->prox;
    p->prox = lixo->prox;
    free (lixo);
}
```

Listas Simplesmente Encadeadas

Buscar e Remover:

Ou pode-se informar o conteúdo a ser excluído. A função deverá buscar a célula correspondente e excluí-la:

Listas Simplesmente Encadeadas

Buscar e Remover:

Ou pode-se informar o conteúdo a ser excluído. A função deverá buscar a célula correspondente e excluí-la:

```
void buscar_e_remover(int y, celula *listaEncadeada){
    celula *p, *q;
    p = listaEncadeada;
    q = listaEncadeada->prox;
    while (q != NULL && q->conteudo != y) {
        p = q;
        q = q->prox;
    }
    if (q != NULL) {
        p->prox = q->prox;
        free (q);
    }
}
```

Listas Simplesmente Encadeadas

```
int main(){
    int resp;
    printf("Lista Simplesmente Encadeada \n");
    celula *topo,*p;

    for(int j=0; j<10;j++){
        printf("Digite o próximo elemento da lista: ");
        scanf("%d", &resp);
        topo = inserir(resp, topo);
        imprimir(topo);
    }
    printf("Fim da Insercao \n");

    buscar_e_remover(5,topo);
    imprimir(topo);
    printf("removeu 5 \n");

    remover(topo->prox);
    imprimir(topo);
    printf("removeu o terceiro \n");
```

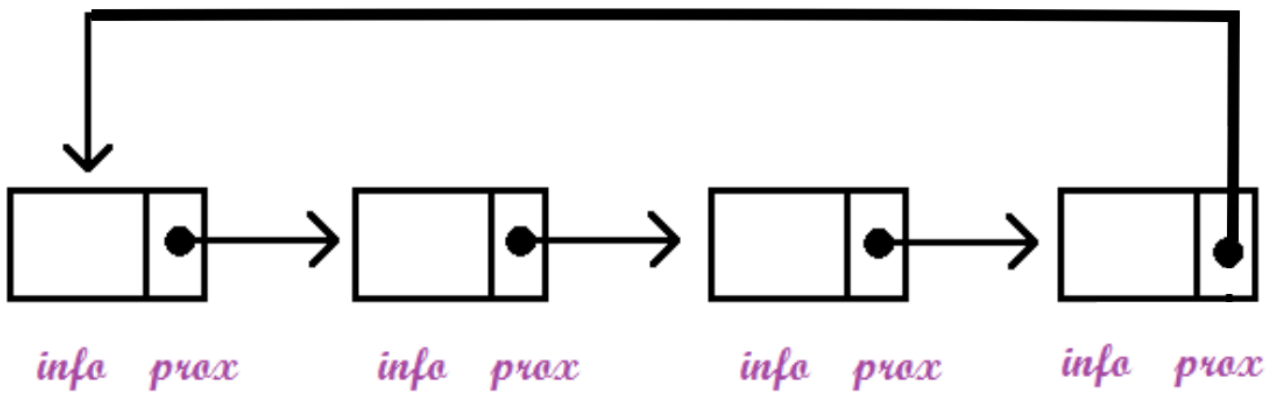
Listas Encadeadas Circulares

Listas Encadeadas Circulares

Uma Lista Encadeada Circular é uma estrutura de dados em que os objetos estão organizados de forma linear, assim como a Lista Simplesmente Encadeada, porém a lista não tem início, meio e fim. Ela pode iniciar de qualquer um dos seus nós. Assim, o último nó apontará para o primeiro, formando um círculo. Os nodos são encadeados de forma unidirecional, pode-se percorrer a lista em apenas um sentido, mas não na direção contrária.
[Cormen et al. 2009]

Listas Encadeadas Circulares

Uma Lista Encadeada Circular é uma estrutura de dados em que os objetos estão organizados de forma linear, assim como a Lista Simplesmente Encadeada, porém a lista não tem início, meio e fim. Ela pode iniciar de qualquer um dos seus nós. Assim, o último nó apontará para o primeiro, formando um círculo. Os nodos são encadeados de forma unidirecional, pode-se percorrer a lista em apenas um sentido, mas não na direção contrária.
[Cormen et al. 2009]



Listas Encadeadas Circulares

Assim como a Lista Simplesmente Encadeada, a Lista Encadeada Circular é uma lista de registros, composta de conteúdo e um ponteiro para o próximo elemento da lista. Em linguagem C, uma lista de *Structs*.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct reg{
    int         conteudo;
    struct      reg *prox;
}celula;
```

Listas Encadeadas Circulares

Inserir:

O elemento pode ser inserido em qualquer posição da lista. O limite do tamanho da lista é o tamanho da memória disponível.

Listas Encadeadas Circulares

Inserir:

O elemento pode ser inserido em qualquer posição da lista. O limite do tamanho da lista é o tamanho da memória disponível.

```
celula *inserir(int x, celula *p){
    celula *nova, *q;
     nova = malloc (sizeof (celula));
     nova->conteudo = x;
     nova->prox = NULL;
    if (p != NULL){
        q = p;
        while(q->prox != p)
            q = q->prox;
        nova->prox = q->prox;
        q->prox = nova;
    } else {
        nova->prox = nova;
    }
    p = nova;
    return p;
}
```

Listas Encadeadas Circulares

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa bem simples:

Listas Encadeadas Circulares

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa bem simples:

```
void imprimir(celula *listaCircular) {
    celula *p, *topo;
    topo = listaCircular;
    for (p = listaCircular; p != NULL; p = p->prox){
        printf ("%d\n", p->conteudo);
        if (p->prox == topo) break;
    }
}
```

Listas Encadeadas Circulares

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse:

Listas Encadeadas Circulares

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse:

```
celula *buscar(int x, celula *listaCircular){
    celula *p, *topo;
    topo = listaCircular;
    p = listaCircular->prox;
    while (p->conteudo != x){
        p = p->prox;
        if (p == topo) break;
    }
    return p;
}
```

Listas Encadeadas Circulares

Remover:

A função remover pode ser construída de uma forma simples, onde passa-se o ponteiro, por valor, do elemento anterior e deseja-se remover o próximo elemento:

Listas Encadeadas Circulares

Remover:

A função remover pode ser construída de uma forma simples, onde passa-se o ponteiro, por valor, do elemento anterior e deseja-se remover o próximo elemento:

```
void remover(celula *p){
    celula *lixo;
    lixo = p->prox;
    p->prox = lixo->prox;
    free (lixo);
}
```

Listas Encadeadas Circulares

Buscar e Remover:

Ou pode-se informar o conteúdo a ser excluído. A função deverá buscar a célula correspondente e excluí-la:

Listas Encadeadas Circulares

Buscar e Remover:

Ou pode-se informar o conteúdo a ser excluído. A função deverá buscar a célula correspondente e excluí-la:

```
void buscar_e_remover(int y, celula *listaCircular){
    celula *p, *q, *topo;
    topo = listaCircular;
    p = listaCircular;
    q = listaCircular->prox;
    while (q->conteudo != y) {
        p = q;
        q = q->prox;
        if (q == topo) break;
    }
    if (q != topo) {
        p->prox = q->prox;
        free (q);
    }
}
```

Listas Encadeadas Circulares

```
int main(){
    int resp;
    printf("Lista Encadeada Circular\n");
    celula *topo,*p;

    for(int j=0; j<10;j++){
        printf("Digite o próximo elemento da lista: ");
        scanf("%d", &resp);
        topo = inserir(resp, topo);
        imprimir(topo);
    }
    printf("Fim da Insercao \n");

    buscar_e_remover(5,topo);
    imprimir(topo);
    printf("removeu 5 \n");

    remover(topo->prox);
    imprimir(topo);
    printf("removeu o terceiro \n");
    return 0;
}
```

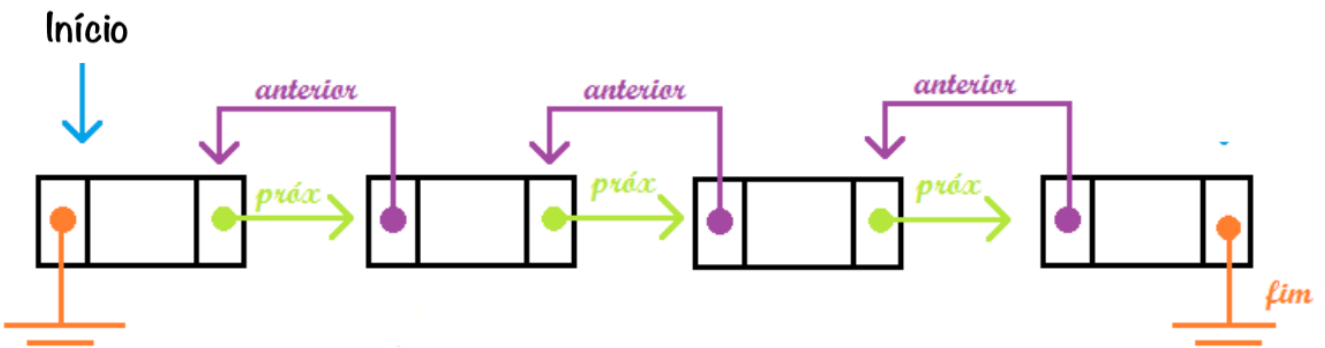
Listas Duplamente Encadeadas

Listas Duplamente Encadeadas

Uma Lista Duplamente Encadeada é uma estrutura de dados semelhante a uma lista simplesmente encadeada, porém além do nó apontar para o próximo nó, ele aponta também para o nó anterior. Assim, permite que a lista seja percorrida em ambos os sentidos. [Cormen et al. 2009]

Listas Duplamente Encadeadas

Uma Lista Duplamente Encadeada é uma estrutura de dados semelhante a uma lista simplesmente encadeada, porém além do nó apontar para o próximo nó, ele aponta também para o nó anterior. Assim, permite que a lista seja percorrida em ambos os sentidos. [Cormen et al. 2009]



Listas Duplamente Encadeadas

Cada nodo, célula ou elemento, da Lista Duplamente Encadeada é composto pelo conteúdo do nodo da lista e por dois ponteiros para o próximo nodo e para o nodo anterior. Assim, uma Lista Duplamente Encadeada é na verdade uma lista de registros. Em linguagem C, uma lista de *Structs*.

Listas Duplamente Encadeadas

Cada nodo, célula ou elemento, da Lista Duplamente Encadeada é composto pelo conteúdo do nodo da lista e por dois ponteiros para o próximo nodo e para o nodo anterior. Assim, uma Lista Duplamente Encadeada é na verdade uma lista de registros. Em linguagem C, uma lista de *Structs*.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct reg{
    int         conteudo;
    struct      reg *prox;
    struct      reg *ant;
}celula;
```

Listas Duplamente Encadeadas

Inserir:

Na Lista Duplamente Encadeada, deve-se tomar cuidado na hora de inserir, pois tanto o ponteiro do elemento anterior, quanto o ponteiro do elemento próximo devem ser manipulados corretamente.

Listas Duplamente Encadeadas

Inserir:

Na Lista Duplamente Encadeada, deve-se tomar cuidado na hora de inserir, pois tanto o ponteiro do elemento anterior, quanto o ponteiro do elemento próximo devem ser manipulados corretamente.

```
celula *inserir(int x, celula *p){
    celula *nova;
     nova = malloc (sizeof (celula));
     nova->conteudo = x;
     nova->prox = NULL;
     nova->ant = NULL;
    if (p != NULL){
         p->ant = nova;
         nova->prox = p;
    }
    p = nova;
    return p;
}
```

Listas Duplamente Encadeadas

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa bem simples. No caso da Lista Duplamente Encadeada, pode ser feita em ordem crescente ou decrescente:

Listas Duplamente Encadeadas

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa bem simples. No caso da Lista Duplamente Encadeada, pode ser feita em ordem crescente ou decrescente:

```
void imprimir(celula *listaDuplaEncadeada) {
    celula *p;
    for (p = listaDuplaEncadeada; p != NULL; p = p->prox){
        printf ("%d\n", p->conteudo);
    }
}
```

Listas Duplamente Encadeadas

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse. Esta função também pode ser construída de forma iterativa ou recursiva e no caso da Lista Duplamente Encadeada, a lista pode ser percorrida em qualquer sentido, vejamos as duas opções:

Listas Duplamente Encadeadas

Buscar:

```
celula *buscar(int x, celula *listaDuplaEncadeada){
   celula *p;
   p = listaDuplaEncadeada;
   while (p != NULL && p->conteudo != x){
      p = p->prox;
   }
   return p;
}

celula *busca_r(int x, celula *listaDuplaEncadeada){
   if (listaDuplaEncadeada == NULL)  return NULL;
   if (listaDuplaEncadeada->conteudo == x)  return listaDuplaEncadeada;
   return busca_r (x, listaDuplaEncadeada->prox);
}
```

Listas Duplamente Encadeadas

Buscar e Remover:

A função remover pode ser construída informando o conteúdo a ser excluído e buscando-o na lista, excluindo a célula correspondente.

Listas Duplamente Encadeadas

Buscar e Remover:

A função remover pode ser construída informando o conteúdo a ser excluído e buscando-o na lista, excluindo a célula correspondente.

```
void buscar_e_remover(int y, celula *listaDuplaEncadeada){
   celula *p, *q;
   p = listaDuplaEncadeada;
   q = listaDuplaEncadeada->prox;
   while ((p != NULL) && (p->conteudo != y)) {
      p = q;
      q = q->prox;
   }
   if (p != NULL) {
      if (q != NULL){
        q->ant = p->ant;
      if (p->ant != NULL)
          p->ant->prox = q;
      free (p);
   }
}
```

Listas Duplamente Encadeadas

```
int main(){
    int resp;
    printf("Lista Duplamente Encadeada\n");
    celula *topo,*p;

    for(int j=0; j<10;j++){
        printf("Digite o próximo elemento da lista: ");
        scanf("%d", &resp);
        topo = inserir(resp, topo);
        imprimir(topo);
    }
    printf("Fim da Insercao \n");

    buscar_e_remover(8,topo);
    imprimir(topo);
    printf("removeu 8, caso existisse \n");

    remover(topo->prox);
    imprimir(topo);
    printf("removeu o terceiro elemento \n");
```

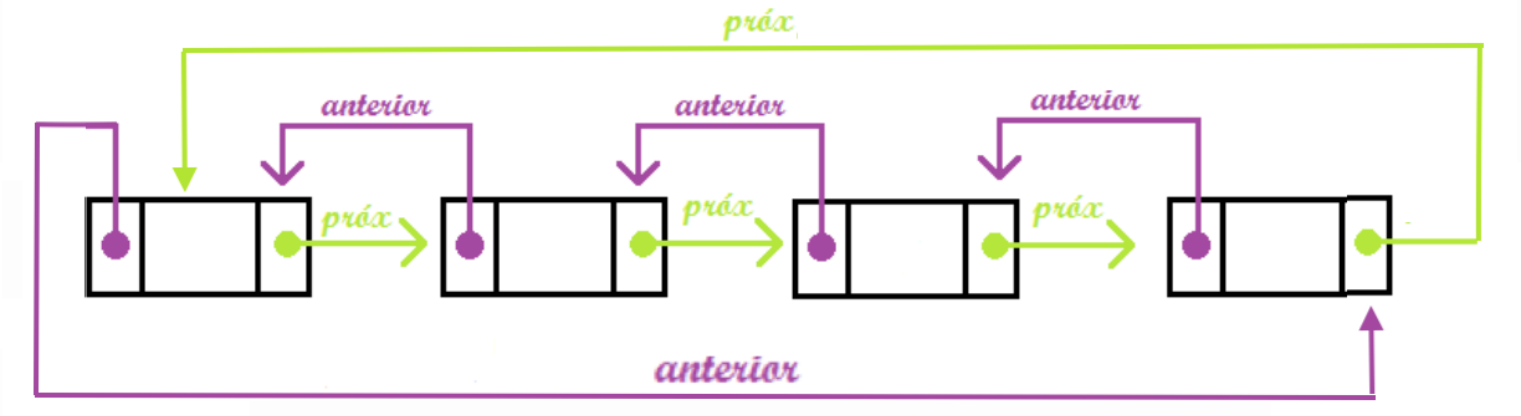
Listas Duplamente Encadeadas Circulares

Listas Duplamente Encadeadas Circulares

Uma Lista Duplamente Encadeada Circular é uma estrutura de dados semelhante a uma lista duplamente encadeada e a uma lista circular. É basicamente uma lista duplamente encadeada cujo último nó aponta para o primeiro, assim não existe início e fim. [Cormen et al. 2009]

Listas Duplamente Encadeadas Circulares

Uma Lista Duplamente Encadeada Circular é uma estrutura de dados semelhante a uma lista duplamente encadeada e a uma lista circular. É basicamente uma lista duplamente encadeada cujo último nó aponta para o primeiro, assim não existe início e fim. [Cormen et al. 2009]



Listas Duplamente Encadeadas Circulares

Cada nodo, célula ou elemento, da Lista Duplamente Encadeada Circular é composto pelo conteúdo do nodo da lista e por dois ponteiros para o próximo nodo e para o nodo anterior. Assim, uma Lista Duplamente Encadeada Circular é na verdade uma lista de registros, assim como as anteriores. Em linguagem C, uma lista de *Structs*.

Listas Duplamente Encadeadas Circulares

Cada nodo, célula ou elemento, da Lista Duplamente Encadeada Circular é composto pelo conteúdo do nodo da lista e por dois ponteiros para o próximo nodo e para o nodo anterior. Assim, uma Lista Duplamente Encadeada Circular é na verdade uma lista de registros, assim como as anteriores. Em linguagem C, uma lista de *Structs*.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct reg{
    int         conteudo;
    struct      reg *prox;
    struct      reg *ant;
}celula;
```

Listas Duplamente Encadeadas Circulares

Inserir:

Na Lista Duplamente Encadeada Circular, deve-se tomar cuidado na hora de inserir, pois tanto o ponteiro do elemento anterior, quanto o ponteiro do elemento próximo devem ser manipulados corretamente e não existe ponteiro nulo, como a lista duplamente encadeada. O primeiro elemento da lista, quando sozinho, aponta o anterior e o próximo para ele mesmo.

Listas Duplamente Encadeadas Circulares

Inserir:

```
celula *inserir(int x, celula *p){
    celula *nova;
    nova = malloc (sizeof (celula));
    nova->conteudo = x;
    nova->prox = nova;
    nova->ant = nova;
   if (p != NULL){
        nova->ant = p->ant;
        p->ant = nova;
        nova->prox = p;
        nova->ant->prox = nova;
   }
   p = nova;
   return p;
}
```

Listas Duplamente Encadeadas Circulares

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa simples. No caso da Lista Duplamente Encadeada Circular, pode ser feita em qualquer ordem, tomando cuidado para não entrar em loop:

Listas Duplamente Encadeadas Circulares

Imprimir:

Exibir os elementos inseridos na lista é uma tarefa simples. No caso da Lista Duplamente Encadeada Circular, pode ser feita em qualquer ordem, tomando cuidado para não entrar em loop:

```
void imprimir(celula *listaDuplaEncadeadaCircular) {
    celula *p, *topo;
    topo = NULL;
    for (p = listaDuplaEncadeadaCircular; p != topo; p = p->prox){
        topo =  listaDuplaEncadeadaCircular;
        printf ("%d\n", p->conteudo);
    }
}
```

Listas Duplamente Encadeadas Circulares

Buscar:

A função buscar consiste em percorrer a lista atrás da célula que contém o conteúdo específico que está sendo buscado. Assim, a função buscar deverá retornar um ponteiro para a célula da lista que contém o conteúdo de interesse. Esta função também pode ser construída de forma iterativa e no caso da Lista Duplamente Encadeada Circular, a lista pode ser percorrida em qualquer sentido:

Listas Duplamente Encadeadas Circulares

Buscar:

```
celula *buscar(int x, celula *listaDuplaEncadeadaCircular){
    celula *p, *topo;
    p = listaDuplaEncadeadaCircular;
    topo = listaDuplaEncadeadaCircular;
    while (p->conteudo != x){
        p = p->prox;
        if (p == topo) break;
    }
    return p;
}
```

Listas Duplamente Encadeadas Circulares

Buscar e Remover:

A função remover pode ser construída informando o conteúdo a ser excluído e buscando-o na lista, excluindo a célula correspondente. Nessa lista, deve-se sempre tomar cuidado para não entrar em loop.

Listas Duplamente Encadeadas Circulares

Buscar e Remover:

A função remover pode ser construída informando o conteúdo a ser excluído e buscando-o na lista, excluindo a célula correspondente. Nessa lista, deve-se sempre tomar cuidado para não entrar em loop.

```
void buscar_e_remover(int y, celula *listaDuplaEncadeadaCircular){
   celula *p, *q, *topo;
   p = listaDuplaEncadeadaCircular;
   q = p->ant;
   topo = listaDuplaEncadeadaCircular;
   while (p->conteudo != y) {
      q = p;
      p = p->prox;
      if (p == topo) break;
   }
   if (p != topo) {
      q->prox = p->prox;
      p->prox->ant = q;
      free (p);
   }
}
```

Listas Duplamente Encadeadas Circulares

```
int main(){
    int resp;
    printf("Lista Duplamente Encadeada Circular\n");
    celula *topo,*p;

    for(int j=0; j<10;j++){
        printf("Digite o próximo elemento da lista: ");
        scanf("%d", &resp);
        topo = inserir(resp, topo);
        imprimir(topo);
    }
    printf("Fim da Insercao \n");

    buscar_e_remover(8,topo);
    imprimir(topo);
    printf("removeu 8, caso existisse \n");

    remover(topo->prox);
    imprimir(topo);
    printf("removeu o terceiro elemento \n");
    return 0;
}
```

Filas

Filas

A Fila consiste numa Estrutura de Dados que funciona obedecendo a ordem FIFO(First In First Out), ou seja, o primeiro a entrar é o primeiro a sair. A Fila pode ser implementada com vetores ou com listas, ou seja com ou sem ponteiros, contanto que obedeça a regra FIFO. Aqui na disciplina usaremos ponteiros em C para implementar nossa Fila.

Filas

As filas são muito utilizadas na computação em diversos campos, como filas de processos em Sistemas Operacionais, filas de pacotes em Redes de Computadores, entre outros exemplos. Então, nenhum profissional de computação deve se eximir de aprender com maestria como manipular uma fila. Na verdade, nenhuma estrutura de dados!

Filas

Basicamente, quando pensamos em filas, pensamos em duas ações importantes: **Enfileira** e **Desenfileira**, ou seja, o elemento chegou e entrará no fundo da fila (enfileira) e o elemento já esperou na fila e agora está no topo e chegou a hora de ser atendido (desenfileira).

Filas

Struct:

A *STRUCT* de uma fila pode ser construída de forma idêntica a uma lista duplamente encadeada.

Filas

Struct:

A *STRUCT* de uma fila pode ser construída de forma idêntica a uma lista duplamente encadeada.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct reg{
    int          conteudo;
    struct       reg *prox;
    struct       reg *ant;
}celula;
```

Filas

Enfileirar:

A função **Enfileirar**, que corresponde a função “inserir” na fila, deve ser realizada de forma que o elemento inserido seja colocado sempre no “FINAL” da fila.

Filas

Enfileirar:

A função **Enfileirar**, que corresponde a função "inserir" na fila, deve ser realizada de forma que o elemento inserido seja colocado sempre no "FINAL" da fila.

```
celula *enfileirar(int x, celula *p){
    celula *nova;
    nova = malloc (sizeof (celula));
    nova->conteudo = x;
    nova->prox = NULL;
    nova->ant = NULL;
   if (p != NULL){
        nova->prox = p;
        p->ant = nova;
   }
   return nova;
}
```

Filas

Desenfileirar:

A função **Desenfileirar**, que corresponde a função “remover” da fila, deve ser realizada de forma que o elemento removido seja removido sempre do “TOPO” da fila.

Filas

Desenfileirar:

A função **Desenfileirar**, que corresponde a função "remover" da fila, deve ser realizada de forma que o elemento removido seja removido sempre do "TOPO" da fila.

```
void desenfileirar(celula *fila){
    celula *p, *q;
    p = fila;
    while (p->prox != NULL)
        p = p->prox;
    q = p->ant;
    if (q != NULL) q->prox = NULL;
    free (p);
}
```

Filas

Buscar:

A função buscar na fila percorre a fila e retorna um ponteiro para o elemento buscado.

Filas

Buscar:

A função buscar na fila percorre a fila e retorna um ponteiro para o elemento buscado.

```
celula *buscar(int x, celula *fila){
    celula *p;
    p = fila;
    while (p!=NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```

Filas

Imprimir:

A função imprimir mostra todos os elementos da fila na ordem do último para o primeiro, como mostrado na figura abaixo.

Filas

Imprimir:

A função imprimir mostra todos os elementos da fila na ordem do último para o primeiro, como mostrado na figura abaixo.

Filas

Imprimir:

A função imprimir mostra todos os elementos da fila na ordem do último para o primeiro, como mostrado na figura abaixo.


```
void imprimir(celula *fila) {
    celula *p;
    for (p = fila; p != NULL; p = p->prox){
        printf ("%d\n", p->conteudo);
    }
}
```

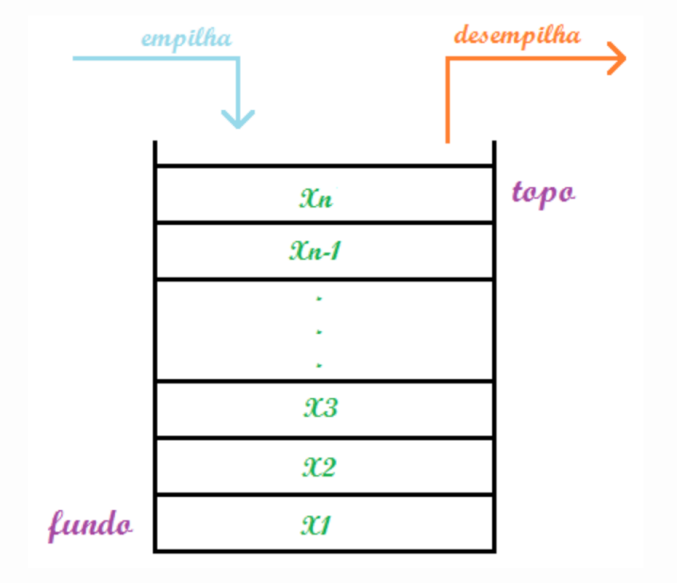
Filas

```
int main(){
    int resp;
    printf("Fila\n");
    celula *fila,*p;

    for(int j=0; j<10;j++){
        printf("Digite o próximo elemento da fila: ");
        scanf("%d", &resp);
        fila = enfileirar(resp, fila);
        imprimir(fila);
    }
    printf("Fim do enfileiramento \n");

    desenfileirar(fila);
    desenfileirar(fila);
    imprimir(fila);
    printf("desenfileirou dois elementos \n");
    return 0;
}
```

Pilhas

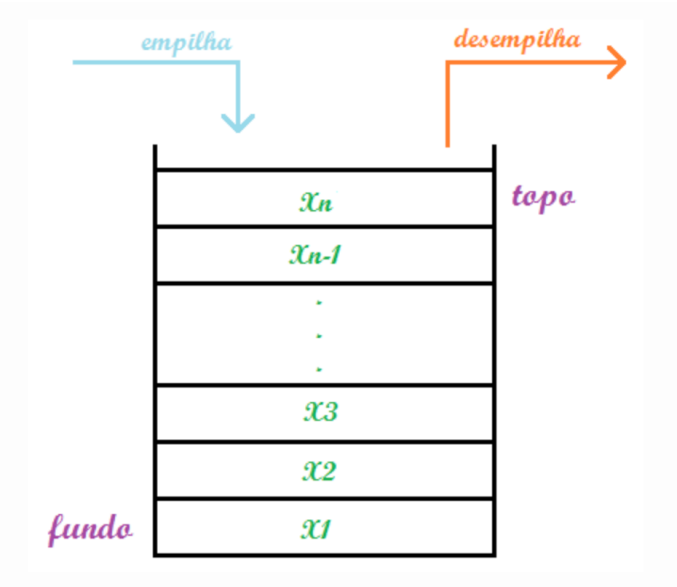
Pilhas

Pilha é uma estrutura de dados que obedece a regra em que os elementos são removidos na ordem do inversa daquela em que foram inseridos de modo que o último elemento que entra é sempre o primeiro a ser retirado, por isto este tipo de estrutura de dados é chamada LIFO (Last In - First Out) ou FILO (First In - Last Out).



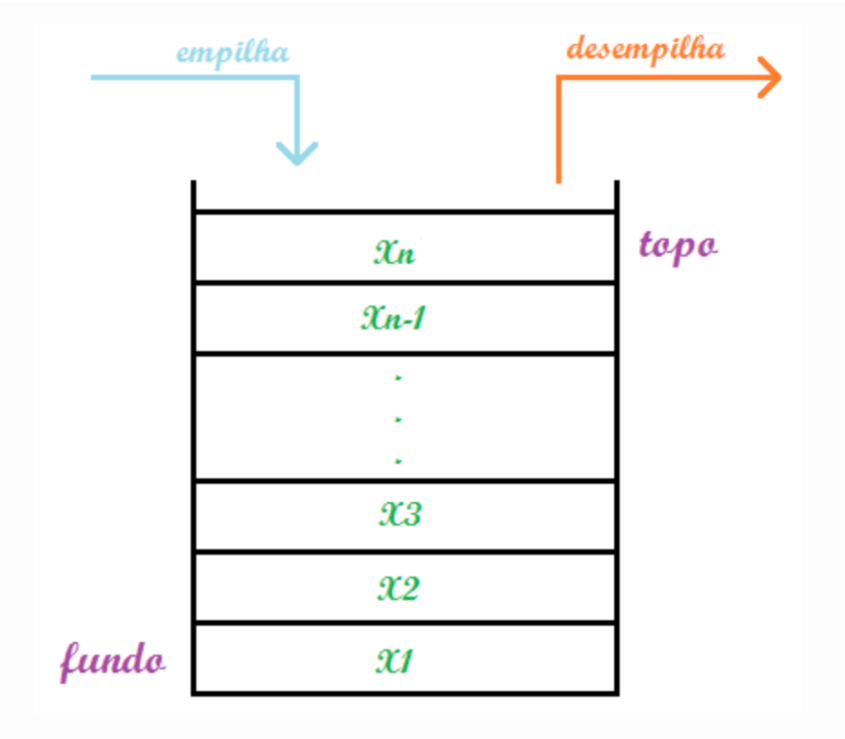
Pilhas

O exemplo mais prático para se entender uma pilha é uma pilha de livros ou pilha de pratos, no qual ao se colocar diversos elementos uns sobre os outros, se quisermos pegar o livro ou o prato mais abaixo deve-se tirar todos os livros, ou pratos, que estão por cima. Outro exemplo é um elevador, a primeira pessoa a entrar num elevador, normalmente, é a última pessoa a sair do elevador.



Pilhas

Basicamente, quando pensamos em pilhas, pensamos em duas ações importantes: **Empilha** e **Desempilha**, ou seja, o elemento chegou e entrará em cima da pilha (empilha) e o elemento será atendido e sairá da pilha (desempilha).



Pilhas

Struct, Imprimir e Buscar:

A *STRUCT*, a função imprimir e a função buscar da Pilha são idênticas às funções da Fila, então não repetiremos elas aqui. Podemos consultar os Slides anteriores ou o Código fonte.

Pilhas

Empilhar:

A função **Empilhar**, que corresponde a função "inserir" na pilha, deve ser realizada de forma que o elemento inserido seja colocado sempre no "TOPO" da pilha.

Pilhas

Empilhar:

A função **Empilhar**, que corresponde a função "inserir" na pilha, deve ser realizada de forma que o elemento inserido seja colocado sempre no "TOPO" da pilha.

```
celula *empilhar(int x, celula *p){
    celula *nova;
    nova = malloc (sizeof (celula));
    nova->conteudo = x;
    nova->prox = NULL;
    nova->ant = NULL;
   if (p != NULL){
        nova->prox = p;
        p->ant = nova;
   }
   return nova;
}
```

Pilhas

Desempilhar:

A função **Desempilhar**, que corresponde a função “remover” da pilha, deve ser realizada de forma que o elemento removido seja sempre retirado do “TOPO” da pilha.

Pilhas

Desempilhar:

A função **Desempilhar**, que corresponde a função "remover" da pilha, deve ser realizada de forma que o elemento removido seja sempre retirado do "TOPO" da pilha.

```
celula *desempilhar(celula *pilha){
    celula *p, *q;
    p = pilha;
    q = pilha;
    if(p != NULL){
        q = p->prox;
        if (q != NULL) q->ant = NULL;
    }
    free (p);
    return q;
}
```

Pilhas

```
int main(){
    int resp;
    printf("Pilha\n");
    celula *pilha,*p;

    for(int j=0; j<10;j++){
        printf("Digite o próximo elemento da Pilha: ");
        scanf("%d", &resp);
        pilha = empilhar(resp, pilha);
        imprimir(pilha);
    }
    printf("Fim do empilhamento \n");

    pilha = desempilhar(pilha);
    pilha = desempilhar(pilha);
    pilha = desempilhar(pilha);
    imprimir(pilha);
    printf("Desempilhou três elementos \n");
    return 0;
}
```

Referências

Referências


CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.